National Institute of Standards and Technology
Computer Systems Laboratory

CHISSA Project Info—
Harvest— Search and Retrieval for CHISSA
WWW Executable Access to CHISSA Tools

September 28, 1995

CHISSA
National Institute of Standards and Technology
Computer Systems Laboratory
Gaithersburg, MD 20899

1. Introduction

''Harvest'' is an integrated set of tools to gather, extract, organize, search, cache, and replicate relevant information across the Internet. With modest effort, ''Harvest'' can be tailored to digest information in many different formats, and offer custom search services on the Internet. ''Harvest'' consists of five subsystems; *Gatherer*, *Broker*, *Replicator* (replicates *Broker*s around the Internet), *Cache* (efficiently retrieve located information), and the *Harvest Server Registry* (holds information about the other ''Harvest'' subsystems on the Internet).

For the current CHISSA needs, only two of the ''Harvest'' subsystems are needed. These subsystems *Gatherer* and *Broker* are described in the following sub-sections.

1.1 ''Gatherer''

The *Gatherer* subsystem, given a set of URLs and associated directives retrieves information using a variety of standard access methods (FTP, Gopher, HTTP, ...). *Gatherer* can retrieve a technical report from an FTP archive, and then extract the author, title, and abstract from the paper to summarize the technical report.

The *Gatherer* consists of a number of separate components.

A configuration file (./harvest-1.3/progs/gatherers/CHISSA/CHISSA.cf) specifies the process of enumerating (the sites to be accessed and files to be retrieved from these sites) and summarizing the data objects (document retrieved). It also provides for customizing (how, when, and how often) the gathering of documents. Password-protected documents can also be gathered by specifying the username and password as part of the URL.

After a document is retrieved, the document is passed through a subsystem called *Essence* to extract indexing information. *Essence*:

- determines the document's data type (e.g., PostScript, tar, ...),

  NOTE: *Essence* recognizes document types in three ways (in order of precedence): by URL naming heuristics, by file naming heuristics, and by locating identifying data within a file using the UNIX *file* command. The type recognition can be controlled by editing file:

  — *lib/byurl.cf to add URL naming heuristics,*

  — *lib/byname.cf to add file naming heuristics, and*

  — *lib/bycontent.cf to add content heuristics.*

- de-archives (*Exploder* used instead of *Summarizer*) the document for nested file types (*uuencoded*, *tar*, *shar*, *compressed*, *gzip*, and *binhex*),

  To customize the unnesting, the file *./src/gatherer/essence/unnest.c* can be altered.

- selects which types of data to index (e.g., do not index Audio files), and

  NOTE: To reject certain files from being indexed, file *lib/stoplist.cf* can be specified. To inform *Essence to index only certain specific types the configuration file lib/allowlist.cf can be specified.*

- applies a type-specific extraction algorithm (called a *summarizer*) to generate a content summary (not performed on nested file types).
- The structured information that the *Gatherer* exports is represented as a list of attribute-value pairs using the *Summary Object Information Format* (SOIF). ‘‘Harvest’’ *Gathers* and *Brokers* communicate using SOIF.

1.1.1 Access to *Gatherer* Information

The information of the *Gatherer* is provided to *Brokers* by the server daemon *gatherd*. Access to the *Gatherer*s database can be controlled by configuration file *gatherd.cf*, which allows specification of Allow or Deny sites.

1.2 “Broker”

The *Broker* subsystem:

- retrieves information from one or more *Gather* subsystems,
- suppresses duplicate information (data from multiple files containing the same data is retrieved only from the first file encountered),
- indexes the collected data, and
- provides a WWW query interface to it.

1.3 Terminology

browsing — the user-guided activity of exploring an information space.

summarizing: — automated process of extracting a representative subset of keywords or other characterizing information from a collection of files.

indexing — automated process of building an efficiently searchable representation of information, which is derived from summarizing.

searching — automated process of traversing an index to respond to a user’s query.

Summary Object Interchange Format (SOIF) — is the designation for representing comment summaries in an attribute-value stream format.

1.4 User Interface

The user would normally interface to ‘‘Harvest’’ via the WWW to access a *cgi-bin* utility. The utility would be personalized to support its project and provide selectable input to assist the user in forming a query for retrieving desired information.

The output from the query can also be tailored to support project desires.

2. Indexing and Summarizing Data

''Harvest'' passes retrieved documents through a subsystem called *Essence*[1].

2.1 *Essence*

Essence summarizes large amounts of data by generating indexes based on an understanding of the semantics of the files indexed. Semantic indexing involves analyzing the structure of file data in different ways depending on file type. By generating information for different types of files in different manners, semantic indexing can generate representative keywords without including every word from a file.

The semantic indexing involves two stages. The classification stage identifies promising files to index and the type information to be associated with each identified file. The summarizing stage applies the appropriate indexing procedure (called a *summarizer*) to each identified file based on the type information uncovered in the classification stage.

Since keyword information is extracted based on knowledge of where high-quality informaion might be located, semantic indexing extracts fewer keywords than full indexing. Yet, it retains the fine-grained associative access capability of full-text indexes.

Essence determines file types by exploiting file naming conventions (such as common filename extensions like ''c''), and locating identifying data or common structures within files (such as UNIX ''magic numbers''). Once the file type is determined, *Essences* executes the appropriate summarizer to extract the keywords from the file.

Essence understands nested file structures, such as compressed, uuencoded ''tar'' files. When *Essence* determines that a file has nested structure, it extracts its files, determines the resulting files' types, and summarizes them. This process continues recursively, until no more nesting is found.

Essence represents content summaries in SOIF. This format preserves file structure information. SOIF provides a means of bracketing collections of summary objects, allowing ''Harvest'' Brokers to retrieve SOIF content summaries from a Gatherer for many objects in a single, efficient compressed stream.

Essence summarizers are autonomous UNIX programs, which are easy to implement, integrate, and maintain. Each summarizer is associated with a specific file type, and understands the file format well enough to extract summary information from the file. The summarizers—Located at *./components/gatherer/standard/summarizers/** in the ''Harvest'' distribution.—are shell scripts. For those file types of interest to CHISSA, the table below provides a summary of the searchable data that *Essence* provides for each file type.

2.2 CHISSA Summarizers

The table below specifies the type of summaries performed for each file type listed. Searchable indexes are generated based only on the summaries obtained for each file.

1. Information on *Essence* can be found from the WWW available documents *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Essence.Jour.ps.Z* and *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Essence.Conf.ps.Z.*

File Type	File’s Text Summarized	UNIX command
Binary	Usage, version, copyright, ManPage summarizer	*strings*
Directory	Extract directory listings, copyright information, and README files.	
HTML	Complete text.	
ManPage	Author, title, etc., based on “-man” macros	
PostScript	Convert to ASCII text	
TeX	TeX to HTML, then HTML summarizer	
Text	First 100 lines & entire README	
Troff	Extract author, title, etc, based on “-man”, “-ms”, “-me” macro packages, or extract section headers and topic sentences.	

2.3 Adding Keywords to CHISSA Documents

To ensure searches based on specific CHISSA topics report a reference to specific documents, the addition of Keywords to CHISSA documents for designated file types was implemented.

2.3.1 Post-Summarizing

“Harvest” supports Post-Summarizing based on URLs. Keywords may be added to the summarized data prior to creating the indexed data. Once added, the document will be selected when the search query is based on an added keyword. The document when retrieved will not contain the added keyword.

2.3.1.1 CHISSA Support

For now, file */export/home/tools/harvest-1.3/CHISSA/MyRules* is designated as the Post-Summarizing configuration file by its entry in */export/home/tools/harvest-1.3/progs/gatherers/CHISSA/CHISSA.cf*. The entry such as:

```
URL == 'http://video/export/home/tools/harvest-1.3/CHISSA/DOCs/F1.html'
        keywords | echo Keyword1 Keyword2
```

adds the keywords *Keyword1* and *Keyword2* to the resultant “Harvest” indexed file for the CHISSA document F1.html.

For each archived file that is to contain a set of keywords, a single URL entry is needed.

When keywords are updated, see the section "Updating search data" (4.2) for the procedures to use for obtaining the proper indexed data.

2.4 WAIS Interface

Support for using WAIS (both free WAIS and WAIS Inc.’s index/search engine) as the Broker’s indexing and search subsystem is included in the “Harvest” distribution. WAIS is an alternative to Glimpse if you need a faster search support and are willing to lose the more powerful query features.

3. Document Search

Broker handles many types of queries. This section specifies some of the queries available.

3.1 Glimpse

“Harvest” supports many different index/search engines, with varying capabilities. At present, the most powerful (and commonly used) search engine is *Glimpse*[2] which supports:

- case-insensitive and case-sensitive queries;
- the ability to match parts of words, whole words, or multiple word phrases (like "resource discovery");
- Boolean (AND/OR) combinations of keywords;
- approximate matches (e.g., allowing spelling errors);
- structured queries (which allow you to constrain matches to certain fields);
- the ability to show matched lines or entire matching records (e.g., for citations);
- the ability to specify limits on the number of matches returned; and
- a limited form of regular expressions (e.g., allowing "wild card" expressions that match all words ending in a particular suffix).

You use the same syntax regardless of what index/search engine is running in a particular Broker, but that not all engines support all of the above features.

The different options - case-sensitivity, approximate matching, the ability to show matched lines versus entire matching records, and the ability to specify match count limits - can all be specified with buttons and menus in the Broker query forms.

A structured query has the form ***tag-name : value***, where *tag-name* specifies the context for interpreting the *value*. Each Broker can support different *tag-name* attributes, depending on the data it holds. The common valid *tag-name*s are:

Abstract	Author	Description	File-Size
Full-Text	Gatherer-Host	Gatherer-Name	Gatherer-Version
Keywords	Last-Modification-Time	Refresh-Rate	Time-to-Live
Title	Type	URL-References	Update-Time

The *value* specifies the search value for the *tag-name* attribute.

Normally, the content summary attributes are made available on request for a particular Broker.

Keyword searches and structured queries can be combined using Boolean operators (AND and OR) to form complex queries. Lacking parentheses, logical operation precedence is based left to right. For multiple word phrases or regular expressions, you need to enclose the string in double quotes, (e.g., "internet resource discovery" or "discov.*").

3.1.1 Simple Queries

The simple keyword search query of ***Arizona*** returns all objects in the *Broker* database containing the word ***Arizona***.

The Boolean query ***Arizona AND desert*** returns all objects containing both words anywhere in the object

2. Additional information on *Glimpse* can be found from the WWW address *http://glimpse.cs.arizona.edu:1994*.

in any order.

The phrase query ***"Arizona desert"*** return all objects containing ***Arizona desert*** as a phrase. Notice that you need to put double quotes around the phrase.

Also, Boolean queries with phrases are allowed. (e.g., ***"Arizona desert" AND windsurfing***)

3.1.2 Structured Queries

The simple Structured query ***Title : windsurfing*** returns all objects where the ***Title*** attribute contains the value ***windsurfing***.

The complex query ***"Arizona desert" AND (Title : windsurfing)*** return asll objects that contain the phrase ***Arizona desert*** and where the ***Title*** attribute of the same object contains the value ***windsurfing***.

3.1.3 Additional *Glimpse* Options

Case insensitive. The query can be made case insensitive (lower case and upper case letters differ).

Keywords can match on word boundaries. Otherwise, a keyword will match part of a word (or phrase). For example, ***network*** will match ***networking***, ***sensitive*** will match ***insensitive***, and ***Arizona desert*** will match ***Arizona desertness***.

Glimpse allows the search to contain a number of errors. An error is either a deletion, insertion, or substitution of a single character. The Best Match option will find the match(es) with the least number of errors. The default is 0 (zero) errors.

NOTE: The previous three options do not apply to attribute names. Attribute names are always case insensitive and allow no errors.

3.1.4 Regular Expressions

Some types of regular expressions are supported by Glimpse. A regular expression search can be much slower that other searches. The following is a partial list of possible patterns. (For more details see the URL "http://glimpse.cs.arizona.edu:1994/glimpse.html" Glimpse manual pages.)

- ***^joe*** matches ***joe*** at the beginning of a line.
- ***joe$*** matches ***joe*** at the end of a line.
- ***[a-ho-z]*** matches any character between 'a' and 'h' or between 'o' and 'z'.
- **.** matches any single character except newline.
- ***c**** matches zero or more occurrences of the character 'c'.
- **.*** matches any number of wild cards.
- ***** matches the character '*'

NOTE: \ escapes any of the above special characters.

Regular expressions are currently limited to approximately 30 characters, not including meta characters. Regular expressions will generally not cross word boundaries (because only words are stored in the index). So, for example, "lin.*ing" will find "linking" or "flinching," but not "linear programming."

3.2 On-line Demonstration

The latest version of "Harvest" (Version 1.3, September 7, 1995) has been installed on "video". Access to a document search demo is obtainable from a WWW Browser via the URL:

http://video/Harvest/brokers/CHISSA/query.html

3.2.1 Search Result Presentation

These checkboxes allow some control of presentation of the query return.

- Display matched lines (from content summaries):
  By selecting this checkbox, the result set presentation will contain the lines of the Content Summary that matched the query. Otherwise, the matched lines will not be displayed. The default is to display the matched lines.
- Display object descriptions (if available):
  Some objects have short, one-line descriptions associated with them. By selecting this checkbox, the descriptions will be presented. Otherwise, the object descriptions will not be displayed. The default is to display object descriptions.
- Verbose display:
  This checkbox allows you to set whether results are displayed listing the filename, host, path, and Content Summary each on separate lines, or just with two lines listing the filename (without a label) and the Content Summary (with a label). The default is verbose.

4. CHISSA Configuration Info

4.1 Setup for CHISSA Document Search

All the documents available for the CHISSA Document Search are contained in */export/home/tools/harvest-1.3/progs/CHISSA/DOCs*. The program ‘‘RunGatherer’’ in directory */export/home/tools/harvest-1.3/progs/gatherers/CHISSA* enumerates and summarizes data objects obtained from the CHISSA Document Search directory. This gathered data is represented as a list of attribute-value pairs using the *Summary Object Interchange Format* (SOIF).

To generate an updated SOIF archive for CHISSA documents:

```
cd /export/home/tools/harvest-1.3/progs/gatherers/CHISSA
./RunGatherer
```

To allow WWW Browser’s to access the CHISSA SOIF archive, a server (the ‘‘Harvest’’ Broker) has to be running on *video*.

To initiate the CHISSA Broker:

```
cd /export/home/tools/harvest-1.3/progs/brokers/CHISSA
./RunBroker
ps -ax | grep broker    # Ensure "glimpse" server active before continuing
```

The current URL to search the CHISSA Document Server is
http://video/Harvest/brokers/CHISSA/query.html.

4.2 Updating search data

Modify configuration file *CHISSA.cf* if needed to include new files.

```
/export/home/tools/harvest-1.3/progs/gatherers/CHISSA
```

Gather the new data.

```
cd /export/home/tools/harvest-1.3/progs/gatherers/CHISSA
rm -rf data          (When gathered files are dropped)
rm -rf tmp           (Should be always performed)
./RunGatherer
```

NOTE: Execute on *video gather localhost 8500* to view the gathered SOIF data.

Update the Broker data to use the just gathered data.

```
cd /export/home/tools/harvest-1.3/progs/brokers/CHISSA
ps -ax | grep broker    # Ensure "glimpse" server active before continuing
kill pid_broker
rm -rf objects
./RunBroker
ps -ax | grep broker    # Ensure "glimpse" server active before continuing
```

NOTE: If two files contain the same data, ‘‘Broker’’ will only report on one of the files.

4.3 File Types

After the ‘‘Gatherer’’ retrieves a document, it passes the document through the *Essence* subsystem to extract SOIF information. *Essence* can determine the type of data pointed to by a URL (e.g., PostScript, HTML, tar, etc) and then applies the type-specific extraction algorithm (called a summarizer) to the data to generate a content summary. ‘‘Harvest’’ is distributed with a stock set of summarizers.

Comments on those types I have tried:

HTML
Works fine.

Troff
Summarizers exist for the Troff macro packages "me", "man", and "ms". "Troff -man" worked fine. "Troff -mgm" (gnu mm macros) resulted in no data being gathered. A summarizer does not exist for "Troff -mgm".

PostScript
Document is gathered, but the data (depending on how the PostScript file packed the data) may not be appropriate for searching.

Text
Worked fine.

4.4 Changing Broker Output

File */export/home/tools/harvest-1.3/progs/cgi-bin/lib* is the configuration file for the Broker Query results. This file appears to be written in HTML for deciphering by Perl. If this is true, should be able to change this file to satisfy specific requests.

4.5 Do I Continue?

4.5.1 “Harvest” support for

“Harvest” seems capable of supporting the document retrieval needs of CHISSA, which are;

1. Storage of documents of various types on host HISSA.
2. WWW access to CHISSA documents.
3. Document search based on user query.
4. Display selected lines of documents that match query.
5. Retrieval of documents based on results of query and user selections.
6. User selectable limit on data results provided.
7. Output display configurable (programmable) for providing icons to depict document types matched.

“Harvest” provides all the above CHISSA needs. The HTML utility of “Harvest” that provides the search and retrieval user interface appears to have most of the CHISSA features desired. Changing the HTML script should be relatively easy. The output comes from Perl scripts and programs. Some Perl knowledge should be attained before changing these scripts.

4.5.2 Specifications/Documents/Feedback Needed

To continue, need:

- Complete specifications of the CHISSA Search and retrieval facility.
- CHISSA documents that constitute the database.
- Feedback on the “On-line Demo”. How should it be changed for CHISSA?

5. WWW Executable Access to CHISSA Tools

5.1 Constraints

Providing access and useability of CHISSA developed tools at NIST/CSL for all users on the WWW independent of the hardware/software used to attain the access was determined to be too broad a scope. Also, to provide access by allowing remote logins was considered dubious because of security considerations.

This scope was reduced to encompass those capabilities needed to process the current set of tools that CHISSA has available for demonstration. These tools used Xwindows to provide the user interface. By requiring the accesser to support Xterm and requiring the user to configure one's system to allow the NIST/CSL host running the CHISSA demonstration tool to establish an Xwindow on one's workstation, the availability and useability of the CHISSA tools becomes attainable.

Access to CHISSA tools can be provided via WWW pages at designated hosts. Useability of the CHISSA tool is via the WWW page spawning a process that runs the selected CHISSA tool.

5.2 It Works

Host "video" was used to develop a WWW page and a program to provide this capability. Once the tool is selected—User invokes a WWW Browser to get to CHISSA's home page.—from a WWW menu page depicting multiple tools, an independent process is spawned to run the selected tool.—The "httpd" server generated process that processed the WWW Browser request exits, after initiating a new process.— The user now has two active windows on one's monitor. One window is the user's local process displaying the WWW Browser output from the CHISSA home page script and the other window displays the initial output from the selected CHISSA tool. Both windows remain active. The CHISSA tool window provides the ability to run the CHISSA tool as though the user was logged in on the NIST/CSL host.

This capability actually allows a user to open as many independent CHISSA tool windows as desired.

Access to a demo of this capability is obtainable via the URL:
http://video/CHISSAdemo/st_demo.html

5.3 Code

The actual code used to provide the WWW executable capability on "video" for the *xforecast* tool is provided below.

```
    /*   Spawn another process to run CHISSA Tool.
      The current Tool run for this cgi-bin module is "xforecast".
    */


#define    _POSIX_SOURCE
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>
```

```
char *conhtml = "Content-type: text/html\n\n";      /* note dual linefeeds */

extern  void est_env(char **);


#define    TRUE    1

extern  char    **environ;

#define    USER_NAME    "cincotta"    /*    This process runs as this user.    */
#define    TESTpath    "/usr/local/bin/xforecast"

#define    REMOTE_HOST    "REMOTE_HOST"
#define    CONSOLE        "console"
    /*    Establish proper environment.    */

/*  Note:           The execution of "xforecast" requires a number of
                    environmental parameters.
*/
#define    LD_LIBRARY_PATH    "LD_LIBRARY_PATH=/usr/local/ORBeline/solaris_gcc/lib:/usr/openwin/lib"
#define    OPENWINHOME        "OPENWINHOME=/usr/openwin"
#define    ORBELINE        "ORBELINE=/usr/local/ORBeline/solaris_gcc/adm"
#define    PATH            "PATH=/usr/local/bin:/usr/ucb:/bin:/usr/bin:/usr/ccs/bin:/usr/local/ORBeline/solaris_gcc/bin:/usr/openwin/bin:/usr/openwin/lib:/usr/ccs/bin:/usr/sbin"
/*  Note:   End of defines for environmental parameters. */


main(int argc, char *argv[], char *env[])
{
    pid_t    pid;
    char    *remote_host;
    char    display[100], console[100];

    est_env(env); /* Establish environment based on WWW browser client data. */

    if(*(getenv(CONSOLE)) == NULL) {    /*    Determine console address.    */
        remote_host = getenv(REMOTE_HOST);
        if(remote_host == NULL)
            _exit(1);
    }
    else
        remote_host = getenv(CONSOLE);

    pid = fork();
    if(pid == (pid_t)0) {
        setuid(200);        /*    Run CHISSA tool as a specific user.    */

            /*    Setup needed environment, which will vary for each tool.  */
        sprintf(display, "DISPLAY=%s:0", remote_host);
        putenv(display);
```

```
/*  Note:   Establish environment needed by the tool. . */
        putenv(LD_LIBRARY_PATH);
        putenv(OPENWINHOME);
        putenv(ORBELINE);
        putenv(PATH);
/*  Note:   End of establishing environment. */

        fclose(stdout);        /*    Required for parent process to exit.    */
        execle(TESTpath, "path_test", (char *)0, environ);
        _exit(1);    /*    Should never get here.    */
    }

/*        Generate WWW Browser display.    */
    fprintf(stdout, "%s", conhtml);
    fprintf(stdout, "<br>");
    fprintf(stdout,    "<TITLE>CHISSA Tool Executed</TITLE>");
    fprintf(stdout, "<hr>");
    fprintf(stdout,    "<H1>CHISSA Tool -<br></H1>");
    fprintf(stdout,    "<br><br><br>");
    fprintf(stdout,    "<H2>");
    fprintf(stdout,    "<pre>       xforecast  <a href=%s>%s",
                "http://video.ncsl.nist.gov/~cincotta/st_demo.html",
                "<img align=middle src=/ICONs/unravel.gif></a></pre>");
    fprintf(stdout, "<br>");
    fprintf(stdout, "has been spawned as an independent process.<br>");
    fprintf(stdout, "<br>");
    fprintf(stdout, "<br>");
    fprintf(stdout, "User's console address:  %s", remote_host);
    fprintf(stdout, "<br>");
    fprintf(stdout, "<hr>");
    fflush(stdout);
    _exit(0);
}
```

As you can see the code needed is minimal. The code can be viewed as consisting of three areas:

1. Place all the inputted parameters provided by the WWW Browser into the environment of the process to facilitate ease of obtaining user specific input (if needed), then spawn a new process for executing the tool.
2. The new process establishes its environment needed to run *xforecast* and the user's monitor as its Xterminal, then executes *xforecast*.
3. Server process establishes a new display for the WWW Browser and exits.

NOTE: To establish the proper display for Xterm the code will use the address of the WWW Browser, UNLESS the user provides the address of one's host. That is, when a user initiates a WWW Browser from other than one's own workstation the user MUST input the proper address of one's workstation.

5.3.1 Generic WWW Tool Execution

The code above executes *xforecast*. To execute another tool, its only the code associated with establishing the proper environment for the tool that may need to be altered. In the code above, the tool environment code is "enclosed" by *Note:* lines.

5.3.2 Tool Execution and Maintenance

To provide maximal expertise for each CHISSA Tool, the point of contact for each tool must be the tool developer. This may not be practical in many cases, but should be the goal for naming the contact point. A single contact for ALL the CHISSA Tools is a poor approach to Tool support.

Since the environment of the tool dictates where the tool can run, not all tools will run on all systems. Therefore, multiple hosts will be needed to run ALL the tools.

To ensure that a CHISSA Tool remains on-line and up-to-date, the tool developer must be an active participant in providing the support of the CHISSA Tool.

By having the Tools runnable on different host sites, distributes the overhead of running the Tools.

5.3.3 Tool Tutorial, My View

Having a tool executable from the WWW requires that the tool either needs no user expertise or that the tool provides a means for the user to attain the skills required to use the tool. There are a number of ways to provide the latter.

- On-line access to a complete report on the tool.
- Buttons that may be selected when help is needed.
- A canned demo.
- An on-line tutorial which runs in concert with the tool.

A complete report should always be available. This allows a user to focus on the details of the tool, its features, and its concepts. Running the tool should not be the only means for deriving a tools use.

For todays Xwindow applications, the tools should always provide a means for providing HELP so that the user may continue. This form of documentation is provided for many of today's major applications. Assistance is only provided when asked. The documentation, in this form, is an integral part of the code for the tool.

A canned demo could also be provided as an option by a tool. The demo would only be run when requested by the user. This allows the user to run the tool without being encumbered with proving the related data and input to run the tool.

Providing an on-line tutorial which runs in concert with the tool in a separate window seems too cumbersome to be useful and too complicated to provide. Having two windows to focus on is one window too many. Multiple windows are very useful for processing, but one's focus is normally concentrated on only one window. Generating the code to determine the data being displayed and updating the other window with its tutorial data seems very ambitious. Also, this would require another program be provided for each tool. A program that would have to be kept in concert with any changes to the real tool program.

6. Current Billing

Job Number 872-7278 Billing

Pay_Week	used	Total_Used	Allocated = 90
16	4	04	
17	40	44	
18	36	80	NOTE: Job# changed to ???

CONTENTS